APPLICATION

FOR

UNITED STATES LETTERS PATENT

TITLE: ERROR CORRECTION FOR MEMORY

APPLICANT: ALPESH B. OZA, MIGUEL A. GUERRERO AND ROHIT R. VERMA

CERTIFICATE OF MAILING BY EXPRESS MAIL

Express Mail Label No. ER847507874US

April 1, 2004

Date of Deposit

Error Correction for Memory

BACKGROUND

When data are stored in a memory device, soft errors can occur that result in data loss or incorrect execution of instructions. One type of error code bits are parity bits that are used to detect the existence of errors but generally cannot correct errors. Another type of error code bits are error correction codes that can be used to detect and correct the errors.

DESCRIPTION OF DRAWINGS

FIG. 1 shows a memory that stores user data, parity information, and error correction information.

FIG. 2 shows a memory controller that implements error correction for read and write operations.

FIG. 3 shows a network processor that includes the memory controller of FIG. 2.

FIG. 4 shows a process for writing data and error correction information into the memory device of FIG. 1.

FIG. 5 shows a process for reading data and, if there is an error, correcting the error to output accurate data.

FIG. 6 shows a memory that stores user data, parity information, and error correction information.

FIG. 7 shows an Ethernet switch blade 300 that includes the network processor of FIG. 3.

DETAILED DESCRIPTION

Referring to FIG. 1, a memory 100 has a first memory region 110 that stores user data and parity information, and a second memory region 112 that stores error correction information. The

user data is stored as 32-bit data words 104, each associated with a 4-bit parity code 106 and a 7-bit error correction code 108. A read instruction causes a 32-bit user data 104 and a 4-bit parity code 106 to be read from the first memory region 110. The parity code 106 is analyzed to determine whether there is error in the user data 104. If there is error, a corresponding 7-bit error correction code 108 is read from the second memory region 112 to correct the error. If there is no error, the error correction code 108 is not read. This allows error correction functionality to be easily implemented in a system designed to access 32-bit data words in the memory using a 36-bit wide data bus (or a 18-bit wide data bus using burst-of-two access, or a 9-bit wide data bus using burst-of-four access).

The term "user data" is used generally to refer to data that is different from the parity information and the error correction information. The parity information is used to determine whether there is error in the user data, and the error correction information is used to correct error, if any, in the user data. The user data can be control data used to control processing of other data, such as network data packets. The user data is sometimes referred to as write data when write operations are involved, and referred to as read data when read operations are involved.

The memory 100 can be a quad-data-rate static random access memory (QDR™ SRAM) that is used in a network processor. Many commercially available QDR SRAM products use data buses that are 9, 18, or 36 bits wide. When a QDR SRAM using a 9-bit data bus is operated using burst-of-four read/write operations, each read/write operation accesses 9x4=36 bits of data. Likewise, when a QDR SRAM using a 18-bit data bus is operated using burst-of-two read/write operations, each read/write operation accesses

18x2=36 bits of data. The 36 bits of data can include 32 bits of user data and 4 bits of parity information.

Referring to FIG. 2, a memory controller 200 interfaces with a commercially available QDR memory 100 and implements error correction functionality by using a write data module 120, a read data module 122, and an ECC correction module 124. The write data module 120 writes write data to the first memory region 110, and writes error correction information to the second memory region 112. Each address (e.g., 126a, 126b, 126c) in the first memory region 110 corresponds to an address (e.g., 128, 130) in the second memory region 112. For example, the error correction code (ECC A0), which is associated with the user data (A0) stored at the address 126a is stored in the first byte at address 128. The error correction code (ECC A1) associated with the user data (A1) stored at address 126b is stored in the second byte at address 128, and so forth.

A 7-bit error correction code can correct an error in a 32-bit user data, and four 7-bit error correction codes can be stored at a single address (e.g., 128 or 130), which corresponds to 36 bits of memory space. In this example, the error correction codes (e.g., ECC A0, ECC A1, ECC A2, ECC A3) stored at address 128 correspond to the user data A0 to A3 stored at addresses 126a to 126d, respectively. The error correction codes (e.g., ECC B0, ECC B1, ECC B2, ECC B3) stored at address 130 correspond to the user data B0 to B3 stored at addresses 127a to 127d, respectively, and so forth.

The write data module 120 receives a 32-bit write data 132 and a write command 134 that requests the write data 132 to be written to a particular address in the memory 100. The write data module 120 calculates an error correction code 136 based on the write data 132 using a conventional error correction code technique, e.g., linear block codes, Hamming codes, or

convolutional codes. The write data module 120 sends the write data 132 and a write command 140 to the memory 100. The write command 140 passes through a multiplexer 146, which multiplexes the write command 140 with other commands and forwards the multiplexed commands 148 to the memory 100. When the memory 100 receives the write data 132 and the write command 140, a parity generator (not shown) in the memory 100 determines a 4-bit parity code (e.g., 160) associated with the write data 132. The write data 132 and the parity code are written into memory at an address (e.g., 126a) designated by the write command 140.

The write data module 120 determines an address for the error correction code based on, for example, a predetermined formula. For example, (address of error correction code) = (address of write data) / 4 + offset, where offset is the starting address of the first error correction code. The write data module 120 generates a byte mask 138 for the error correction code 136 and sends the byte mask 138, the error correction code 136, and another write command 162 to the memory 100. The write command 162 includes an address (e.g., 128) in the memory region 112 where the error correction code 136 is to be stored.

The byte mask 138 determines which byte at the address (e.g., 128) is to store the error correction code. For example, the write data in address 126a is associated with the error correction code ECC A0, which is stored in the first byte at address 128. In this example, the byte mask 138 masks off the 2^{nd} to 4^{th} bytes at the address 128, so that the ECC is written into the 1^{st} byte. Likewise, the write data 132 in address 127b is associated with the error correction code ECC B1, which is stored in the 2^{nd} byte at address 130. In this example, the byte mask 138 masks off the 1^{st}, 3^{rd}, and 4^{th} bytes at the address 130, so that the ECC is written into the 2^{nd} byte.

In the example above, two instruction cycles are used to write a 32-bit data into the memory 100. In the first cycle, the write command 140 causes the write data to be written into the memory region 110. In the second cycle, the write command 162 causes the error correction code to be written into the memory region 112.

The memory controller 200 includes a read data module 122 that receives a read command 150 that requests data to be read from a particular address in the memory 100. The read data module 122 sends a read command 142 to the memory 100 (through the multiplexer 146), which causes a 32-bit read data 152 and the associated 4-bit parity code 164 to be read from the memory region 110. The read data module 122 determines whether there is an error in the read data 152 based on the parity code 164. If there is no error, the read data 152 is passed to an ECC correction module 124 and output as read data 158.

If the read data module 122 determines that there is error based on the parity code 164, the read data module 122 instructs the ECC correction module 124 to send an ECC read command 144 to the memory 100 (through the multiplexer 146). This causes an error correction code 156 to be read from the memory 100. The ECC correction module 124 corrects the error in the read data 152 based on the error correction code 156 and outputs the read data 158, which has been corrected for error.

The write data 132, the write command 134, and the read command 150 may be sent from, for example, a data processor (not shown). The read data 158 may be sent to the data processor.

The memory controller 200 is useful in implementing error correction functionality in a system designed to access 32 bits of data per read/write operation. The write data module 120 requires two instruction cycles to write data and error correction information into the memory 100. The read data module

122 reads the error correction information only when there is error in the read data 152 (as determined based on the parity information 164). Because soft errors occur infrequently, the percentage of read operations that require a subsequent reading of the error correction information is low. In a system where a read operation is performed more frequently than a write operation, the memory controller 200 provides error correction functionality while having a small impact on data access throughput.

Referring to FIG. 3, the memory controller 200 can be used in a network processor 210 that is used to process data packets sent over a network. In this example, the network processor 210 has four QDR SRAM memory controllers 200 to control access to QDR SRAM devices. The four memory controllers 200 allow access to four independent memory channels to provide sufficient control information bandwidth for 10-gigabit network applications. SRAM devices are useful for storing control information, which tends to have many small data structures such as queue descriptors and linked lists. SRAM devices allow for small access size, and allow for access to an arbitrary address sequence. In one example, each memory controller 200 runs at 200 MHz, provides 800 MB/s of read bandwidth and 800 MB/s of write bandwidth.

The memory controllers 200 also provide the following functions: (1) Atomic read-modify-write operations, such as increment, decrement, add, subtract, bit-set, bit-clear, and swap. (2) Linked-list queue operations, such as enqueue and dequeue to linked-list operations. (3) Ring operations, including inserting data at the tail of a ring or removing data from the head of the ring.

The network processor 210 includes microengines 280 that process data packets. There are 16 microengines 280 that are

configured as two clusters of eight microengines to provide more communication capability between the microengines and the remainder of the processor resources.

An embedded processor 270 handles exception packets and performs management tasks. A media and switch fabric interface 272 is used to connect the network processor 210 to a physical layer device and/or a switch fabric. The switch fabric interface 272 includes a receive buffer 274 to receive incoming packets and a transmit buffer 276 to hold outgoing packets.

A bus system 278 interconnects the units within the network processor 210, and includes data busses that connect the microengines 280 to various shared resources, such as SRAM, dynamic random access memory (DRAM), hash, and cryptography units, described below.

Three independent DRAM controllers 282 control external DRAM devices, and allow access to three independent channels to provide sufficient data buffering bandwidth for 10-gigabit network applications. DRAM devices are useful for data buffering because they offer good burst bandwidth and can be denser and cheaper per bit relative to SRAM devices.

Cryptography units 284 perform authentication and bulk encryption. A hash and scratch unit 286 implements hash functions to detect collisions, and has a small on-chip scratchpad memory, which functions similarly to an external SRAM device, but with a lower latency.

A PCI unit 288 provides an interface to a PCI data bus 290. The PCT unit 288 can be used to interface with an external control microprocessor, or be used to interface with an external device, such as a public key accelerator. The PCI unit can act as a PCI bus master, allowing the embedded processor 270 or microengines 280 to access external PCI targets, or as a PCI bus target, allowing external devices to transfer data to and from

the external SRAM and DRAM devices coupled to the network processor 210.

Referring to FIG. 4, a process 230 for writing data and error correction information to the memory 100 is shown. The write data module 120 in the memory controller 200 receives (232) a write command and write data from a data processor. The write data module 120 generates (234) error correction code based on the write data. The write data module 120 sends (236) a first write command and the write data to the memory 100. A parity generator in the memory 100 generates (238) a parity code associated with the write data. The write data and the associated parity code are stored (240) at a first address in the first memory region 110. The write data module 120 sends (242) a second write command and the error correction code to the memory 100. The error correction code is stored (244) at a second address in the second memory region 112.

FIG. 5 illustrates a process 250 for reading data from the memory 100 and correcting error, if any, in the read data. The read data module 122 in the memory controller 200 receives (252) a read command. The read data module 122 sends the read command to the memory 100, causing read data and associated parity code to be read (254) from the memory 100. The read data module 122 determines (256) whether there is error in the read data based on the parity code. If there is no error, the read data is output (258) to the data processor. If there is error, the error correction module 124 determines (260) the address of the error correction code that is associated with the read data. The error correction module 124 reads (262) the error correction code from the memory 100, and uses the error correction code to correct (264) the error in the read data. The corrected read data is output (258) to the data processor.

Referring to FIG. 6, in a memory 220, the user data and the error correction codes are interleaved so that four 32-bit data words are followed by four error correction codes. For example, user data is stored at addresses 0x0001 to 0x0004, followed by error correction information stored at address 0x0005, and user data stored at addresses 0x0006 to 0x0009 is followed by error correction information stored at address 0x000A. The memory controller 200 maps virtual addresses (e.g., addresses recognized by a data processor) to physical addresses (i.e., actual addresses in the memory 220). For example, virtual addresses 0x0001 to 0x0004 map to physical addresses 0x0001 to 0x0004, respectively, and virtual addresses 0x0005 to 0x0008 map to physical addresses 0x0006 to 0x0009, respectively, and so forth. When the user data (e.g., A0) is read from an address (e.g., 0x0001), the associated error correction code (e.g., ECC A0 at address 0x0005) is not read unless there is error in the user data based on the parity code.

The user data 104 can have lengths different from 32 bits, the parity codes can have lengths different from 4 bits, and the error correction codes can have lengths different from 7 bits. Memory 100 is not limited to QDR SRAM, and can be other types of memory, such as conventional SRAM, MoSys® 1T-SRAM, conventional DRAM, fast page mode DRAM, extended data out (EDO) DRAM, burst EDO DRAM, pipeline burst EDO DRAM, synchronous DRAM, enhanced synchronous DRAM, virtual channel DRAM, reduced latency DRAM, fast cycle DRAM, Direct Rambus DRAM, etc.

Referring to FIG. 7, an example of a 10-gigabit Ethernet switch blade 300 includes a network processor 210a for ingress processing and a network processor 210b for egress processing. Ingress processing tasks may include classification, metering, policing, congestion avoidance, statistics, segmentation, and traffic scheduling into a switch fabric. Egress processing tasks

may include reassembly, congestion avoidance, statistics, and traffic shaping. Input and output buffering are supported using buffers in DRAM devices 292 linked together by linked lists maintained in SRAM devices 304.

The ingress and egress network processors 210a, 210b present a full-duplex interface to a switch fabric through interface chip 302. Packets are streamed into the ingress network processor 210a at or above line rate. The packets are received, reassembled, processed, buffered into the DRAM devices 292, and enqueued for transmission into the switch fabric. Similarly, the egress network processor 210b receives packets from the fabric interface 302, reassembles, processes, and buffers the packets in the DRAM devices 292, and queues the packets for outgoing transmission through an egress framer 296.

In one example, a DRAM interface 298 includes three Rambus™ DRAM channels operating at a clock rate of up to 533 MHz, offering an aggregate peak bandwidth of 51 Gb/s. An SRAM interface 304 includes four QDR II SRAM channels that may be clocked up to 250 MHz. Each interface supports an independent read and write port, providing an aggregate read rate of 32 Gb/s and, simultaneously, an aggregate write rate of 32 Gb/s. This configuration provides a cost-effective and flexible approach to basic packet processing at 10 Gb/s rates.

Although some examples have been discussed above, other implementations and applications are also within the scope of the following claims. For example, the locations for storing error correction information may be arranged differently from what is shown in FIG. 1 and FIG. 6. The memory controller 200 can operate in different speeds and can be used in different types of machines.